Computing the Gale-Shapley Algorithm in R: An Introduction to matchingR

Jan Tilly

May 17, 2015

Introduction

This package provides R functions to quickly compute stable matchings for large matching markets using the Gale-Shapley Algorithm (Gale and Shapley 1962). This package can be useful when the number of market participants is large or when very many matchings need to be computed (e.g. for extensive simulations or for estimation purposes). The package has successfully been used to simulate preferences and compute the matching with 30,000 participants on each side of the market. The package supports one-to-one, one-to-many, and many-to-one matchings.

Matching markets are very common in practice and widely studied by economists. Popular examples include

- the National Resident Matching Program that matches recent graduates from medical school to residency programs at teaching hospitals throughout the United States
- the matching of students to schools including the New York City High School or the the Boston Public School Match (and many more)
- the matching of kidney donors to recipients in kidney exchanges.

This package implements the the Gale-Shapley Algorithm to compute a stable matching for such markets.

Consider the following example of the marriage market. Suppose there are `N` men and `N` women. A matching assigns men to women such that each man is assigned to exactly one woman and each woman is assigned to exactly one man. A matching is **stable** if there is no man `m` that would rather be matched to some other woman `w'` than to his current spouse `w` and this other woman `w'` would rather be matched to man `m` than to her current spouse `m'`. In other words, a matching is stable when there does not exist a pair `(m, w')` for which both `m` and `w'` are better off than they are with their respective spouses `w` and `m'`.

Preferences

We first need to introduce notation to specify men's and women's preferences. Men's payoffs from being matched are given by the payoff matrix `uM`. Columns in the matrix correspond to men, rows to women. For example:

```
uM = matrix(c(1.0, 0.5, 0.0,
              0.5, 0.0, 0.5,
              0.0, 1.0, 1.0), nrow = 3, ncol = 3, byrow = TRUE)
```

```
##        cols
## rows    Woman 1 Woman 2 Woman 3
##   Man 1     1.0     0.5     0.0
##   Man 2     0.5     0.0     0.5
##   Man 3     0.0     1.0     1.0
```

In this example, man 1 receives a payoff of 1.0 from being matched to woman 1, a payoff of 0.5 from being matched to woman 2 and a payoff of 0.0 from being matched to woman 3 (same logic applies to men 2 and 3). Similarly, let the payoff matrix for women be given by

```
uW = matrix(c(0.0, 1.0, 0.0,
              0.5, 0.0, 0.5,
              1.0, 0.5, 1.0), nrow = 3, ncol = 3, byrow = TRUE)
```

```
##          cols
## rows      Man 1 Man 2 Man 3
##   Woman 1   0.0   1.0   0.0
##   Woman 2   0.5   0.0   0.5
##   Woman 3   1.0   0.5   1.0
```

Here, columns in the matrix correspond to women, rows to men. In this example, woman 1 receives a payoff of 0.0 from being matched to man 1, a payoff of 0.5 from being matched to man 2 and a payoff of 1.0 from being matched to man 3 (same logic applies to women 2 and 3).

Instead of using payoff matrices, we can also represent preferences using preference orderings. The preference ordering that corresponds to uM is

```
prefM = matrix(c(1, 3, 3,
                 2, 1, 2,
                 3, 2, 1), nrow = 3, ncol = 3, byrow = TRUE)
```

```
##        cols
## rows    Woman 1 Woman 2 Woman 3
##   Man 1       1       3       3
##   Man 2       2       1       2
##   Man 3       3       2       1
```

prefM states that man 1 prefers woman 1 over woman 2 over woman 3, etc. The preference ordering that corresponds to uW is given by

```
prefW = matrix(c(3, 1, 3,
                 2, 3, 2,
                 1, 2, 1), nrow = 3, ncol = 3, byrow = TRUE)
```

```
##          cols
## rows      Man 1 Man 2 Man 3
##   Woman 1     3     1     3
##   Woman 2     2     3     2
##   Woman 3     1     2     1
```

The matching algorithm discussed below can take either payoff matrices of the type uM and uW or preference orderings of the type prefM and prefW as arguments.

Gale-Shapley Algorithm: One-to-one matching

The Gale–Shapley algorithm works as follows: Single men ("the proposers") sequentially make proposals to each of their most preferred available women ("the reviewers"). A woman can hold on to at most one

proposal at a time. A *single* woman will accept any proposal that is made to her. A woman that already holds on to a proposal will reject any proposal by a man that she values less than her current match. If a woman receives a proposal from a man that she values more than her current match, then she will accept the proposal and her previous match will join the line of bachelors. This process continues until all men are matched to women.

For the preferences specified in `uM` and `uW`, we can compute the Gale-Shapley Algorithm by hand. Initially, all men are single.

1.
   - Man `1` proposes to woman `1`, his most-preferred choice.
   - Unmatched men: `2`, `3`
2.
   - Man `2` proposes to woman `3`, his most-preferred choice.
   - Unmatched men: `3`
3.
   - Man `3` proposes to woman `3`, his most-preferred choice.
   - Woman `3` now dumps man `2`.
   - Unmatched men: `2`
4.
   - Man `2` proposes to woman `1`, his most-preferred available choice.
   - Woman `1` now dumps man `1`.
   - Unmatched men: `1`
5.
   - Man `1` proposes to woman `2`, his most-preferred available choice.
   - All men are now matched.

The man-optimal stable matching is thefore:

Man 1	Woman 2
Man 2	Woman 1
Man 3	Woman 3

The package computes the Gale-Shapley algorithm using the function `one2one`:

```
matching = one2one(uM, uW)
```

Note that we can obtain equivalent results when we use `prefM` and `prefW` as arguments:

```
matching = one2one(proposerPref = prefM, reviewerPref = prefW)
```

The function `one2one` returns a list `matching` that includes the vectors `proposals` and `engagements` with the final proposals and engagements, respectively. These two vectors contain the same information (i.e. they tell us who is matched with whom). For the example above, the vector of proposals contains

```
matching$proposals
```

```
##        cols
## rows    Proposed to Woman
##   Man 1                 2
##   Man 2                 1
##   Man 3                 3
```

The first element in the vector tells us that man 1 is matched with woman 2. Man 2 is matched to woman 1, and man 3 is matched to woman 3. The vector of engagement contains

```
matching$engagements
```

```
##          cols
## rows      Engaged to Man
##   Woman 1              2
##   Woman 2              1
##   Woman 3              3
```

The first element in the vector tells us that woman 1 is matched to man 2, woman 2 will be matched to man 1, and woman 3 will be matched to man 3.

We can then check if the computed matching is stable using the function `checkStability`. To check if a matching is stable, we check for each assignment `(m,w)` if there is some other woman `w'` that man `m` would rather be matched with and who would rather be matched to man `m`. This function will return `true` if the matching is stable and `false` otherwise.

```
checkStability(uM, uW, matching$proposals, matching$engagements)
```

For the simple 3-by-3 example, we can perform this check by hand:

- Man 1 is matched to woman 2, his second-most preferred choice. His most preferred choice is woman 1. Woman 1 is matched with man 2 who she prefers over man 1. Thus man 1 cannot do better than woman 2.
- Man 2 is matched to woman 1, his second-most preferred choice. His most preferred woman is woman 3, who is matched with man 3. Since man 3 is her most-preferred choice, man 2 cannot do better than woman 1.
- Man 3 is matched to women 3, his most preferred choice, so he cannot do better.

Thus, this matching is stable.

Extensions

The following examples illustrate some additional features of this package.

Marriage Market with Unequal Number of Men and Women

The following is an example of `one2one` with different numbers of participants on each side of the market. There are 2,500 women and 2,000 men. By construction, 500 men will remain unmatched. We randomly generate payoff matrices `uM` and `uW` which are drawn from a uniform distribution (`runif`). We then compute the male-optimal (i.e. men are proposing) and the female-optimal (i.e. woman are proposing) matching.

```
# set seed
set.seed(1)
# set number of men
nmen = 2500
# set number of women
nwomen = 2000
# generate preferences
```

```
uM = matrix(runif(nmen*nwomen), nrow = nwomen, ncol = nmen)
uW = matrix(runif(nmen*nwomen), nrow = nmen, ncol = nwomen)
# male optimal matching
resultsM = one2one(uM, uW)
# female optimal matching
resultsW = one2one(uW, uM)
# check if matchings are stable
checkStability(uM, uW, resultsM$proposals, resultsM$engagements)
checkStability(uW, uM, resultsW$proposals, resultsW$engagements)
```

College Admissions Problem: One-to-Many Matching

The following is an example of `one2many` where 1000 students get matched to 400 colleges, where each college has two slots. By construction, 200 students will remain unmatched. We draw students' and colleges' preferences, `uStudents` and `uColleges`, respectively, by from a uniform distribution.

```
# set seed
set.seed(1)
# set number of students
nstudents = 1000
# set number of colleges
ncolleges = 400
# generate preferences
uStudents = matrix(runif(ncolleges*nstudents), nrow = ncolleges, ncol = nstudents)
uColleges = matrix(runif(nstudents*ncolleges), nrow = nstudents, ncol = ncolleges)
# worker optimal matching
results = one2many(uStudents, uColleges, slots = 2)
# check if matching is stable
checkStability(uStudents, uColleges, results$proposals, results$engagements)
```

Literature

Gale, David, and Lloyd S Shapley. 1962. "College Admissions and the Stability of Marriage." *American Mathematical Monthly*, 9–15.